Institut für Softwaretechnologie

Abteilung Software Engineering

Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Bachelorarbeit Nr. 93

Spreadsheet-Fehlermuster

Sebastian Beck

Studiengang:	Softwaretechnik B.Sc.
Prüfer:	Prof. Dr. S. Wagner
Betreuer:	Daniel Kulesz, M.Sc.
Begonnen am:	01.11.2013
Abgeschlossen am:	03.05.2014
CR-Klassifikation:	D2.4, H4.1

Zusammenfassung

In den Achtzigern begann eine weitgehende Verbreitung von Spreadsheets in der Industrie als Folge davon, das Spreadsheet-Software für den PC erschienen ist, mit der auch nicht-programmiererfahrene Endnutzer Spreadsheets erzeugen und bearbeiten konnten. Heutzutage sind sie in fast allen Unternehmen nicht mehr weg zu denken. Jedoch zeigen viele Studien, dass ein Großteil dieser Spreadsheets Fehler enthalten und dadurch jährlich hohe Schäden verursachen. Obwohl Prüfwerkzeuge für Spreadsheets existieren, sind sie auf Grund von entwurfstechnischen Einschränkung in der Industrie nur in geringem Maße einsetzbar. Deshalb wurde an der Universität Stuttgart in bisher je 2 abgeschlossenen Diplom- und Bachelorarbeiten das Spreadsheet Inspection Framework erstellt. Das Ziel des Spreadsheet Inspection Frameworks ist es durch statische und dynamische Prüfungen, die selber erstellt und konfiguriert werden können, auf Fehler im Spreadsheet aufmerksam zu machen.

In dieser Arbeit wurden Fehlermuster entwickelt um die statischen Prüfungen des Spreadsheet Inspection Frameworks zu erweitern. Das Fehlermuster „Einer unter Anderen“ erkennt Schemata und davon abweichende Zellen finden. „Separate Mehrfach-referenzierung in Formeln“ deckt Fehler beim Erstellen von Funktionen auf. „Eingaben an nicht berücksichtigten stellen“ erkennt sowohl falsche als auch fehlende Referenzierungen. Zudem werden Werte erkannt, die in falsche Zellen gesetzt wurden. Das Fehlermuster „Referenzierung auf null-Werte“ durchsucht das Spreadsheet nach Referenzierungen, die sich auf eine leere Zelle beziehen. Das „Wortdistanz“ Muster sucht mit Hilfe des Levenshtein Algorithmus nach Tippfehlern.

Summary

In the 1980s began a wide dissemination of spreadsheets in the industry as a result of the release of spreadsheet-software which allow end-users without programming experience to create and handle spreadsheets. It's inconceivable for most companies to work without spreadsheets. Many studies show that most of these spreadsheets contain errors and as a consequence there are high damages per year. Although there are tools for spreadsheets, they are rarely used in the industry because of conceptual restrictions. Because of that the Spreadsheet Inspection Framework was developed at the University Stuttgart as part of two finished diploma thesis and two finished bachelor thesis. The objective of the Spreadsheet Inspection Framework is it to red-flag errors with the help of static and dynamic tests, which can be developed and configurated on their one.

In this Thesis smell patterns were developed to enhance the static inspections of the Spreadsheet Inspection Framework. The smell pattern "Einer unter Anderen" recognizes schemata and cells which goes against it. „Separate Mehrfachreferenzierung in Formeln" shows errors at the creating of formulas. „Eingaben an nicht berücksichtigten stellen" recognizes false and missing references. Furthermore it perceives values which were set into a false cell. The smell pattern „Referenzierung auf null-Werte" searches for references that point to a null value. The "Wortdistanz" pattern searches on the basis of the Levenshtein algorithm for typing errors.

Inhaltsverzeichnis

1. Einführung

1.1. Motivation

Als die erste Tabellenkalkulationssoftware für den Personal Computer, VisiCalc, erschienen ist, wurden Spreadsheets für Nutzer ohne Programmierkenntnis verwendbar. Dies verhalf Spreadsheets dazu sich in allen Wirtschaftszweigen zu verbreiten. Heutzutage sind sie aus fast keinem Unternehmen mehr weg zu denken, da sie direkt von Endanwender erstellt werden können. Dabei reicht die Spanne der erstellten Spreadsheets von einfachen, kleinen Spreadsheets um kurze Zwischenberechnungen zu machen bis hin zu sehr komplexen und umfangreichen Spreadsheets auf denen unternehmerische Entscheidungen getroffen werde.

Zahlreiche Studien[Butler00, CHM08, LL04, Panko98], die unter anderem in der Praxis verwendete Spreadsheets untersucht haben, zeigen, dass über 80% dieser Spreadsheets Fehler enthalten. Obwohl dieser Mangel an Qualität gut dokumentiert ist und es kommerzielle Werkzeuge gibt, die den Entwickler des Spreadsheets bei der Fehlersuche unterstützen, wird in vielen Unternehmen die Prüfung der Spreadsheets nur unzureichend oder überhaupt nicht umgesetzt. Dies ist zum Teil der verfügbaren Prüfwerkzeuge geschuldet, da diese meist nur einige, einfache Fehlermuster erkennen. Weiterhin ist ihr Programmcode oftmals nicht offen einsehbar, geschweige denn erweiterbar oder konfigurierbar auf die Anforderungen eines Unternehmens. Um diesen Aspekt zu verbessern wurde an der Universität Stuttgart im Rahmen einer Diplomarbeit [Zitze12] ein Framework entwickelt, das „Spreadsheet Inspection Framework“, das in einer Diplom- [Lemke13] und 2 Bachelorarbeiten [Doust13, Scheu14] erweitert wurde. Das Spreadsheet Inspection Framework ist darauf ausgelegt, dass man es gut um weitere Prüfmuster erweitern kann. Zudem sind die bereits implementierten und auch zukünftigen Fehlermustererkennungen konfigurierbar um sie an das jeweilige Einsatzgebiet anzupassen. Da jedoch erst 3 Fehlermuster implementiert wurde und diese nur qualitative Aspekte eines Spreadsheets prüfen, soll das Spreadsheet Inspection Framework um weitere, vor allem qualitative Fehlermuster erweitert werden.

1.2. Ziele

Ziel dieser Arbeit ist es diese statischen Fehlermuster für das Spreadsheet Inspection Framework zu entwickeln. Sie sollen auf Grundlage von in realen Spreadsheets entdeckten Fehlern entwickelt. Die Fehlermustererkennungen sollen dabei konfigurierbar und zum Teil daraufhin entwickelt werden, dass sie Fehlermuster finden, die am häufigsten für Fehler in realen Spreadsheets verantwortlich sind.

Des Weiteren soll gezeigt werden, dass das Spreadsheet Inspektion Framework anhand der neu implementierten Fehlermuster mit anderen Spreadsheet-Prüfwerkzeuge konkurrieren kann. Dies soll validiert werden durch den Vergleich mit dem SmellSheet Detective, der im Rahmen einer wissenschaftlichen Arbeit [CFMMS, CFRS12] entstanden ist.

1.3. Beitrag

Im Verlauf dieser Arbeit wurden anhand von manuell gefunden Fehlern und Berichten über Fehler in Spreadsheets 5 Fehlermuster erarbeitet und konzipiert. Die konzipierten Fehlermuster „Referenzierung von null-Werten“ und „Wortdistanz“ wurden in das Spreadsheet Inspection Framework implementiert. Die bei Erstellung der Ausarbeitung nicht implementierten Fehlermuster wurden auf Grundlage des Programmcodes beschrieben um eine Implementierung zu vereinfachen.

1.4. Übersicht

Diese Arbeit gliedert sich in folgende Kapitel:

1 – Einführung: Hier werden die Motivation hinter der Arbeit und ihre Ziele erläutert. Ebenso wird ein Überblick über die Kapitel gegeben

2 – Grundlagen: In diesem Kapitel werden Grundlagen zu Spreadsheets und dem Spreadsheet Inspection Frameworks vermittelt.

3 – Spreadsheet-Fehler: An dieser Stelle werden auf Fehler in Spreadsheets, Definition eines Fehlers, einer Taxonomie zur Einteilung von Fehlertypen und Arten von Prüfverfahren.

4 – Related work: An dieser Stelle wird ein kurzer Überblick auf nahverwandte Arbeiten gegeben und diese kritisch bewertet.

5 – Konzept: Hier werden die Gründe und die Konzepte der Fehlermuster beschrieben. Es wird anhand von Beispielen gezeigt, wie es zu Fehlern kommen kann und wie versucht wird diese Fehler statisch mit Hilfe von Fehlermustern zu erkennen.

6 – Umsetzung: In diesem Kapitel wird auf die Implementierung der in Kapitel 5 erarbeiteten Fehlermuster näher eingegangen. Zudem wird auf Feinheiten bei der Konfigurierung eingegangen.

7 – Fazit: An dieser Stelle wird eine Zusammenfassung der Arbeit gegeben.

2. Grundlagen

Im diesem Kapitel werden Grundlagen vermittelt, die notwendig sind für das Verständnis und der Erarbeitung der in Kapitel 4 und Kapitel 5 vorgestellten Spreadsheet-Fehler und Fehlermuster. Das Kapitel gliedert sich dabei in Grundlagen zu Spreadsheets, indem Aufbau und Begrifflichkeiten zu Spreadsheets erklärt werden und einem kurzen Überblick über das Spreadsheet Inspection Framework.

2.1. Spreadsheets

Um Fehlermuster beschreiben zu können, werden im Folgenden Grundbegriffe für Spreadsheets erläutert. Dabei wird besonders auf die zum Verständnis wichtigen Begriffe eingegangen. Die Definitionen sind angelehnt an die in der Arbeit von S. Zitzelsberger [Zitze12] verwendeten Definitionen.

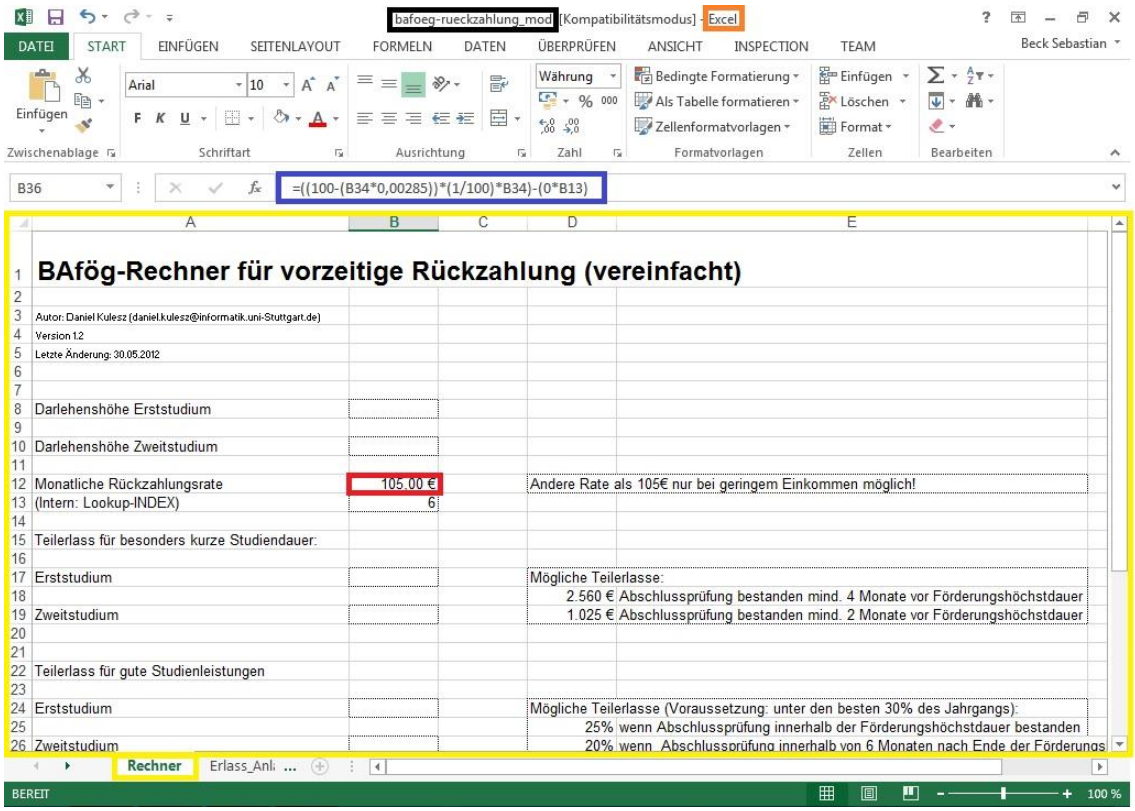

Abbildung 1: Ein Bafög-Spreadsheet in Excel2013

In Abbildung 1 sind folgende Elemente durch ihre Farbcodierung gekennzeichnet:

Spreadsheet, Spreadsheet-Software, Worksheet, Zelle, Formel.

Ein Spreadsheet bedient sich des Grundkonzepts einer zweidimensionalen, rechtwinklig angeordneten Tabelle. Es beinhaltet Daten, Formeln, Referenzierungen und Darstellungsinformationen. Diese Informationen werden in einer Spreadsheet-Sprache implementiert und von einem Spreadsheet-Programm in Form einer oder mehrerer Worksheets elektronisch dargestellt.

Worksheets sind die elektronische Version eines Spreadsheets. Worksheets innerhalb eines Spreadsheet müssen dabei eindeutige Namen haben. Dadurch lassen sich Referenizierungen über verschieden Worksheet hinweg innerhalb eines Spreadsheets realisieren. Zum Beispiel um in Abbildung 1 im Worksheet „Erlass_Anlage“ eine Referenz auf B12 von „Rechner“ zu machen, müsste man in der entsprechenden Zelle „=Rechner!B12“ eingeben.

Eine Spreadsheet-Software ist eine Entwicklungsumgebung für Spreadsheets. Sie beherrscht dabei eine oder mehrere Spreadsheetsprachen, wodurch sie Spreadsheets erstellen, interpretieren, modifizieren und ausführen kann. Das Spreadsheet wird dabei elektronisch als Worksheets dargestellt. Bekannte Vertreter von Spreadsheet-Software sind unter anderem Microsoft Excel und Apache OpenOffice Calc.

Eine Spreadsheetsprache beinhaltet eine Menge von Konstruktionen, um ein Spreadsheet zu beschreiben. Spreadsheetsprachen können unterschiedliche Konstrukte dazu verwenden, stellen jedoch alle die Grundfunktionen, um das Spreadsheetkonzept zu realisieren, bereit.

Eine Zelle ist die kleinste Einheit in einem Spreadsheet. Sie kann Konstanten oder Formeln beinhalten. Konstanten sind in Form von nummerischer Werte oder Strings repräsentiert. Auf Formeln wird im Weiteren genauer eingegangen. Zellen werden über den Spaltenbuchstaben gefolgt von der Zeilenzahl genau identifiziert. Die rotmarkierte Zelle in Abbildung 1 hätte somit die Zelladresse B12.

Ein Bereich ist ein rechteckiger Zusammenschluss von Zellen. Dabei wird auf die rechte obere und die untere linke Ecke des Bereiches verwiesen. Eine Referenzierung auf einen solchen Bereich würde wie folgt aussehen: „=Summe(B4:C6)“

Eine Spalte beinhaltet alle Zellen auf einer vertikalen Linie. Spalten werden mit Großbuchstaben alphabetisch durchnummeriert.

Eine Zeile beinhaltet alle Zellen auf einer horizontalen Linie. Zeilen werden nummerisch durchnummeriert beginnend mit der 1.

Eine Referenzierung ist der Bezug auf den Inhalt einer anderen Zelle. Dieser wird durch die Nennung der Zelladresse, auf die Bezug genommen werden soll, vorgenommen. Ist diese auf einem anderen Worksheet wird der Name des Worksheets plus ein Ausrufezeichen vorangestellt, zum Beispiel „=Rechner!B6“.

Bei der Referenzierung werden zwei Formen unterschieden: die ***relative*** und die ***absolute*** Referenzierung.

Bei der relativen Referenzierung wird auf den Wert Bezug genommen, der beim Erstellen an dieser Position stand. Wird die Referenzierung in eine andere Spalte kopiert, ändern sich die Bezüge auf das Umfeld der neuen Spalte.

Bei der absoluten Referenzierung wird jedoch auf eine bestimmte Zelladresse Bezug genommen. Ändert sich der Wert in dieser Zelle, ändert sich der Bezug mit. Dies wird durch ein vorangestelltes Dollarzeichen spezifiziert. Somit würde eine absolute Referenzierung für B12 wie folgt aussehen: „=$B12“.

Eine Formel ist eine Operation, die den Wert genau einer Zelle anhand einer oder mehrerer Operanten berechnet ohne dabei Auswirkungen auf benutzte Referenzen zu haben. Eine Formel besteht also aus ***Operanden***, ***Operatoren*** und ***Funktionen***.

Operanden sind hierbei Konstanten, Referenzierungen oder Ergebnisse von Berechnungen.

Als Operatoren werden am öftesten mathematische Operatoren verwendet, zum Beispiel zum Addieren („+“) oder Vergleichen („<, >, =“).

Funktionen stellen Berechnungen bereit, die zum Teil nicht durch einzelne Operatoren realisierbar sind. Es gibt zum Addieren mehrerer Zahlen zum Beispiel „=SUMME(B3:B7)“, die alle Zahlen der Zellen B3 bis B7 addieren. Darüber hinaus gibt es aber auch komplexere Berechnungen als Funktionen wie Durchschnittswerte und Standardabweichungen.

2.2. Spreadsheet Inspection Framework

Das Spreadsheet Inspection Framework bietet in seiner derzeitigen Ausbaustufe die Möglichkeit statische und dynamische Prüfungen durchzuführen und sich die Ergebnisse in einem Excel2013 Add-in namens SIFEI darstellen zu lassen.

Der Ablauf einer Prüfung wird durch den Aufruf von Scan unter dem Reiter „INSPECTION“ eingeleitet. Dabei kann eine dynamische, eine statische oder eine gesamte Prüfung durchgeführt werden. Zuvor erstellte Szenarien liefern Testwerte für die dynamische Prüfung.

Das SIFEI sendet nach der Einleitung eines Prüfverfahrens das Spreadsheet und einen xml-String, der die Prüfvorschriften enthält, über eine Sockel-Verbindung an das Spreadsheet Inspection Framework (SIF). Im SIF werden die dynamischen Prüfungen anhand der in dem xml-String definierten Prüfvorschriften geprüft. Die statischen Prüfungen werden für alle in der Main-Klasse von SIF registrierten Fehlermuster durchgeführt. Der daraus entstehende Bericht mit den gefunden Befunden wird über die Sockel-Verbindung zurück an das SIFEI geschickt und dort visualisiert.

3. Spreadsheet-Fehler

In diesem Abschnitt wird aufgezeigt, dass es zu Fehlern in Spreadsheets kommt und erläutert, warum dies passiert. Im Folgenden geht es um die Definition von Fehlern. Im Anschluss wird eine Fehlertaxonomie für Spreadsheets aufgezeigt, um dann zu erläutern wie man Fehler innerhalb eines Spreadsheets finden kann.

3.1. Fehler in Spreadsheets

In Kapitel 1.1 wurde anhand von Studien gezeigt, dass die meisten in der Praxis eingesetzten Spreadsheets Fehler enthalten. Eine in der Studie [Panko98] referenzierte Untersuchungen von Lukasic aus dem Jahre 1998, die Spreadsheets auf Fehlerquoten untersucht, zeigt, dass bei den untersuchten Spreadsheets jede fünfzigste Zelle Fehler enthielt. Sie untersucht nur 2 Spreadsheets, lässt jedoch auf eine vergleichsweise hohe Fehlerquote im Durchschnitt von mindestens über 0,5% der Zellen in Spreadsheets schließen. Dieser Schluss wird durch die Arbeit von R. R. Panko [Panko98] unterstützt, in der eine Studie, die eine Fehlerquote von 0,4% entdeckt, die kritisch gesehen wird und der Fehler vorgeworfen werden, da er diese Fehlerquote als zu niedrig ansieht. Dieser Missstand ist gut dokumentiert, es gibt Berichte [EuSpRIG] über hohe finanzielle und reputative Schäden auf Grund von Spreadsheetfehlern und es wird versucht Lösungen in Form von Fehlerfindungsmethoden anzubieten. Dies führt jedoch nicht zu merkbaren Verbesserungen der Situation. Einerseits wird die Suche nach Fehlern in Spreadsheets noch nicht die Aufmerksamkeit geschenkt wie zum Beispiel bei der Softwareentwicklung, wo Prüfen heute zum Standard gehört. Ohne festgesetzte Richtlinien in Unternehmen zur Durchführung von Spreadsheet-Prüfungen, werden Prüfungen weiterhin, wenn sie überhaupt gemacht werden, nur sehr unstrukturiert und dadurch wenig effektiv durchgeführt. Durch die fehlenden Strukturen werden Prüfungen meist von dem Entwickler des Spreadsheets selbst erledigt. Da aber das Vertrauen in die eigene Arbeit meist hoch ist, sind diese Prüfungen meist nicht sehr wirkungsvoll, was auch in verschiedenen Studien gezeigt wird. Ebenso sind die unterstützenden Prüfwerkzeuge nur mit einer überschaubaren Anzahl an Fehlermustern ausgestattet, wodurch der Benutzer bei der Suche von vielen Fehlern nicht unterstützt wird. Abschließend wird festgehalten, dass Fehler in Spreadsheets bekannt und gut dokumentiert sind, jedoch die Unterstützung beim Prüfen sowohl vom Unternehmen als auch von Prüfwerkzeugen unzureichend ist.

3.2. Fehlerdefinition

Am Anfang ist zu klären, was genau ein Fehler ist und wie die umgangssprachliche Verwendung von Fehler im Deutschen in einem Spreadsheet Verwendung findet.

Ein Fehler wird vermutet wenn der Ist-Wert vom Soll-Wert abweicht. Man kann an dieser Stelle bei Spreadsheets nur von einer Vermutung sprechen, da es nicht zwingend Soll-Ergebnisse gibt beziehungsweise diese aus dem Spreadsheet abgeleitet sind, da diese Berechnungen erst durch Hilfe von Spreadsheet gemacht wurden. In so einem Fall spricht man von einem Orakelproblem (darauf wird in 4.4 noch genauer eingegangen). Wird jedoch ein Befund angezeigt, wird er im Allgemeinen als Fehler beziehungsweise error bezeichnet. Das Wort Fehler wird dabei zu allumfassend verwendet. Der IEEE Standard zur Klassifizierung von Software-Anomalien [IEEE09] definiert Begriffe um Fehler besser zu beschreiben. Dazu werden die Begriffe *error*, *fault*, *failure* und *problem* genutzt, die im Folgenden mit deutschen Äquivalenten versehen werden, die im Anschluss in dieser Arbeit verwendet werde. Die Ursache eines Fehlers, also das Fehlverhalten des Erstellers eines Spreadsheets, wird als *error* bezeichnet. Der dadurch entstehende Fehler im Spreadsheet wird hier als *fault* bezeichnet. Hier trifft das deutsche Wort Fehler dann zu. Dieser Fehler wird aber nur durch Abweichungen von dem spezifizierten Verhalten erkannt. Das kann unter anderem an Abweichungen vom Soll-Wert liegen oder daran, dass die Berechnungen nicht ausgeführt werden. Dieses Abweichen ist dabei der *failure*. In der weiteren Arbeit wird für diese Abweichung vom spezifizierten Verhalten das Wort Befund benutzt. Am Schluss dieser Kette steht das *problem*. Es ist so gesehen die Auswirkung des Fehlerverhaltens.

Da es, wie oben angesprochen, sein kann das die Soll-Werte falsch sind beziehungsweise nicht bestimmt werden können, ist nicht jeder Befund auch ein Fehler. Des Weiteren kommt es zu einem Befund, wenn das Fehlermuster anschlägt. Dies ist jedoch nicht gleichbedeutend mit einem Fehler, sondern nur eine Abweichung von einer zu überprüfenden Norm. Deshalb müssen Befunde noch einmal untergliedert werden im Bezug darauf, ob sie ein Fehler sind.

- falsch-positive Befunde haben bei der Prüfung gegen die Norm beziehungsweise den Soll-Wert („positiv“) Differenzen aufgezeigt, sind jedoch keine Fehler („falsch“).
- richtig-positive Befunde haben bei der Prüfung ebenso gegen den Soll-Wert beziehungsweise die Norm verstoßen sind jedoch tatsächlich auch Fehler.

In dieser Taxonomie gibt es auch noch die Begriffe falsch-negativ und richtig-negativ. Falsch-positiv bezeichnet hierbei Fehler, die nicht von der Norm beziehungsweise dem Soll-Wert abgewichen sind, somit nicht erkannt wurden, aber dennoch Fehler sind. Falsch-negativ bezeichnet dabei das keine Fehler gefunden wurde, es an dieser Stelle aber auch keine gab Diese Taxonomie ist abgeleitet aus der Arbeit von Ludewig und Lichter [LuLi07].

3.3. Fehlertaxonomie bei Spreadsheets

Eine Taxonomie ist ein Klassifikationsschema. Bei einer Fehlertaxonomie werden alle Fehler in Klassen und Unterklassen hierarchisch eingeteilt. Dies ist sehr wichtig um einzelne Fehler zuzuordnen und auf Grundlage dieser Einordnung kann man Fehlermuster besser beschreiben.

In der Arbeit [Panko96] von Panko und Halverson wurde eine Fehlertaxonomie entwickelt, die oft genutzt und auf dessen Grundlage viele weitere entstanden sind. Jedoch wurde diese auf der Forschungsbasis der neunziger Jahre entwickelt. Da viele die auf diese Taxonomie verweisen, jedoch auf Grundlage von fehlenden Aspekten auf weitere andere Studien Bezug nehmen müssen um auch neue Erkenntnisse auf diesem Forschungsgebiet abzubilden, hat Panko diese Taxonomie überarbeitet und angepasst [Panko09]. Im Folgenden wird diese Taxonomie erklärt anhand deren die Fehler in dieser Arbeit eingeordnet wurden.

Gegenüber der alten Taxonomie werden erstmal alle Fehler unterteilt in Violations, sprich absichtliche Zuwiderhandlung von Vorschriften bei der Spreadsheetentwicklung und Errors. Errors werden in diesem Kontext jedoch nicht negativ gesehen. Es wird davon ausgegangen, dass Fehler passieren und der Entwickler, solang dies nicht mit Vorsatz geschehen ist, dafür nicht verantwortlich gemacht wird.

Errors werden weiter unterteilt in qualitative und quantitative Fehler. Ein quantitativer Fehler zeichnet sich dadurch aus, dass daraus ein falsches Ergebnis erzeugt wird. Nun könnte man denken das qualitative Fehler zwar unschön, aber nicht so gefährlich sind, da diese im Umkehrschluss keine falschen Ergebnisse liefern. Das Problem bei qualitativen Fehlern ist, dass sie zum einen die Wartbarkeit und Erweiterbarkeit verschlechtern, aber auch direkt zu quantitativen Fehlern führen können. Als Beispiel könnte man hier ein falsch beziehungsweise schlecht platziertes Label ansehen. Die Berechnung und ihre benutzten Referenzierungen stimmen eventuell, aber der Anwender setzt den Wert als Resultat einer für ihn nicht sichtbaren Fehlinterpretation des Labels an eine falsche Stelle. Dann hat ein qualitativer Fehler einen quantitativen Fehler ausgelöst.

Die Quantitativen Fehler werden dann weiter unterteilt in Slips/Lapses und Mistakes. Slips/Lapses sind hervorgegangen aus den mechanischen Fehler aus der alten Taxonomie. In einer von Panko selber erstellten Studie, in der 82 Probanden je ein Spreadsheet entwickeln sollten, wurden die Fehler in die damalige Taxonomie eingeteilt und 41% der Fehler wurden mechanischen Fehlern zugeteilt. Da diese auf die neue Taxonomie übertragen werden können, kann man davon sprechen das Slips/Lapses fast die Hälfte aller Fehler ausmachen. Deshalb konzentrieren sich einige Fehlermuster dieser Arbeit auf diese Kategorie. Slips sind dabei motorische beziehungsweise sensorische Fehler, wie Tipp- und Zeigefehler. Lapses sind Fehler die aus falschen Erinnerungen an den richtigen Wert entstehen.

Mistakes wird in 4 weitere Untergruppen aufgeteilt:

- Domain: Fehler beim Verständnis der Domäne.
- Logic: Fehler in der Logik, zum Beispiel bei der Erzeugung eines Algorithmus.
- Math: mathemische Fehler.
- Software: Fehler bei der Verwendung der Software, zum Beispiel bei der Nutzung von eingebauten Funktionen.

Diese Taxonomie wurde in Abbildung 3, die anhand der Arbeit [Panko09] erstellt wurde, visualisiert.

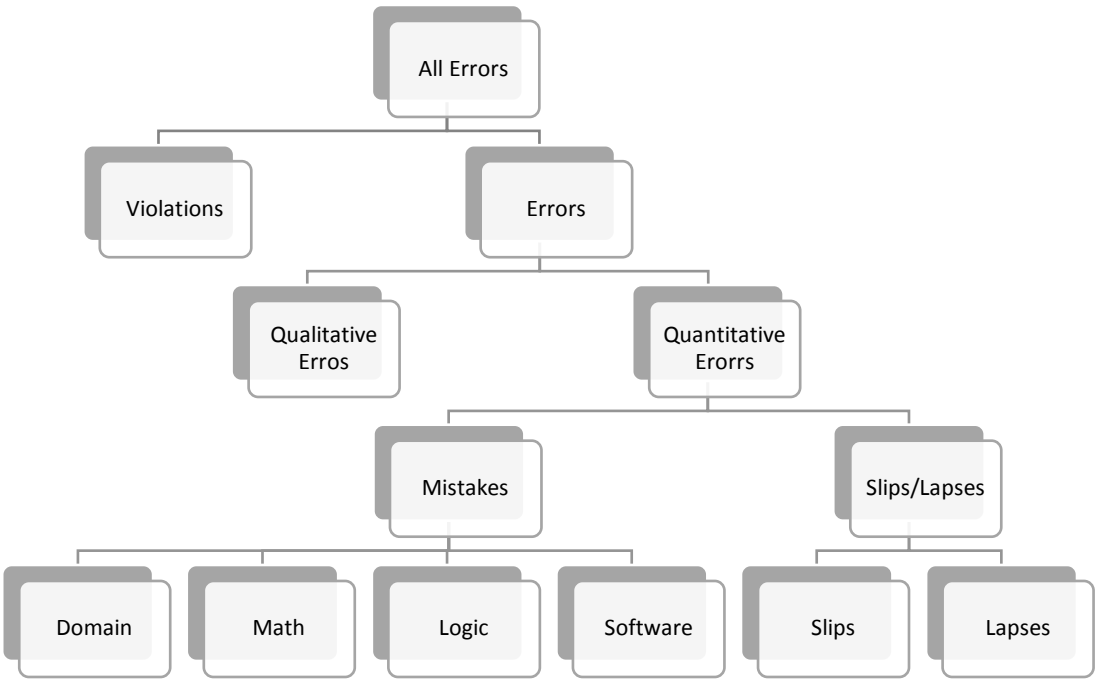

Abbildung 3: Fehlertaxonomie nach Panko [Panko09].

3.4. Prüfverfahren

Das Prüfen von Spreadsheets kann man grob in zwei große Felder einteilen: Mechanische und nicht-mechanische Prüfverfahren.

3.4.1. Nicht-mechanische Prüfung

Bei der einfachsten Form des **nicht-mechanischen** Prüfens wird das Prüfsubjekt vom Ersteller oder einem Kollegen manuell durchgeschaut. Dies ist als einziges Prüfverfahren bei Spreadsheets, das in der Industrie in einem gewissen Umfang vertreten ist. Dieses Verfahren wird meist ungeplant durchgeführt, da entsprechende Richtlinien in den Unternehmen fehlen. Dementsprechend werden laut einer Studie [GHJJR96] weniger als 50% der Fehler, die in Spreadsheets enthalten sind, gefunden. In der Studie [Panko07] wird zu einem formalen Ansatz ähnlich einer Code Inspection im Software-Engineering geraten. Dabei sollen Spreadsheets geplant, im Kreise von mehreren Entwicklern

zusammen zeilenweise durchgeschaut werden und nach zuvor definierten Richtlinien bewertet werden. Die Durchsicht eines Spreadsheets von mehreren Personen gegenüber einer Einzelperson zeigt, dass zumindest im Rahmen eines Experiments mehr Fehler gefunden werden [Panko09].

3.4.2. Mechanische Prüfung

Bei mechanischen Prüfungen wird das Prüfsubjekt mit Hilfe von Software geprüft. Hierzu werden in dieser Arbeit auch Prüfverfahren gezählt, die neben dem mechanischen Teil der Prüfung auch einen nicht-mechanischen Teil besitzen.

Bei der mechanische Prüfung gibt es zwei generelle Ansätze: Die statische und die dynamische Prüfung.

Bei der statischen Prüfung wird der Prüfling nicht ausgeführt. Er wird statisch untersucht und der Prüfling wird dabei nicht verändert. Es gibt verschiedene Ansätze um eine statische Prüfung durchzuführen.

- Man kann nach Mustern suchen, die auf einen Fehler hindeuten. Dabei wird ein Fehlermuster für einen bestimmten Fehlertypen erstellt und so formuliert, dass Vorschriften entstehen gegen die der Prüfling geprüft werden kann. Dieses Konzept wird in vielen Arbeiten[CFRS12, HPD11] eingesetzt um Fehler zu finden, da gut gegen feststehende Vorschriften geprüft werden kann und damit viele Fehlerbereiche erfasst werden können
- Man kann den Prüfling analysieren um nachzuvollziehen, wie der Datenfluss innerhalb des Prüflings funktioniert. In einem Spreadsheet könnte somit Referenzierungsketten dargestellt werden, die in sehr langer Form zu qualitativen Mängeln führen kann oder durch unbeachtete Änderung an anderen Stellen zu falschen Referenzierungen. Dieses Konzept folgt der Arbeit von Taylor [TO80] zu Datenfluss.
- Man kann den Prüfling nach Wiederholungen von Strukturen untersuchen. Im Bezug auf Spreadsheets könnte man hier mehrmals gleiche Formeln entdecken, die, falls sie die gleiche Berechnung vornehmen, ausgelagert werden sollten. Ansonsten kann es zu Fehlern kommen, wenn die Formel bei Änderungen nicht an allen vorkommenden Stellen abgeändert wird. Das Grundprinzip der Entdeckung von sich wiederholenden Strukturen wird in der Arbeit [RD04] erläutert.

Bei der dynamischen Prüfung wird der Prüfling ausgeführt. Dabei werden Testeingaben gemacht und die Ergebnisse mit Soll-Werten verglichen. Hier liegt aber auch ein grundlegendes Problem. Es kann zu einem Orakel Problem kommen, das heißt dass die Korrektheit von Soll-Werten nicht bestätigt werden kann. Um dieses Problem zu umgehen gibt es verschiedene Ansätze.

- Man kann den Endanwender als Testorakel verwenden. Durch das fachspezifische Wissen des Anwenders kann dieser eine objektive Ergebnisbewertung abgeben und sagen, ob ein solcher Wert plausibel ist. Er kann dabei zwar oftmals nur ein Intervall angeben in dem das Ergebnis liegen sollte, jedoch kann dies beim Auffinden von schweren Fehlern helfen.
- Metamorphic testing setzt sich nicht als Ziel die Korrektheit eines Szenarios, sprich einer Eingabemenge mit der passenden Ausgabe, sicherzustellen. Bei dieser Art des Testens wird versucht, dass die Relation zwischen Eingabewerten und Ausgabewerten gleichbleibend ist. Zuerst werden, wie auch beim Testen von Programmcode, die Szenarien in Gruppen eingeteilt, wobei jede Gruppe einen übergeordneten Testfall abdeckt. Ein Repräsentant jeder Gruppe wird dann dynamisch getestet. Ab diesem Punkt setzt dann das metamorphic testing(MT) ein. Die Funktionsweise des MT wird dabei in einer Studie anhand eines einfachen Beispiels wie folgt gezeigt: Gibt es einen Testfall mit sin(49°) wäre ein passender metamorphischer Testfall dazu sin(49° + 360°). Da bei trigonometrischen Funktionen meist nur Näherungswerte angezeigt werden, wird hier der Umstand ausgenutzt, dass die Addition von 360° am Ergebnis nichts ändern darf. Ändert sich jedoch der Ausgabewert, so zeigt dieser Testfall, dass es wahrscheinlich einen Fehler im Spreadsheet gibt. Dies ist Grundlage der Arbeit [PKLC13] von Poon et al..

4. Related work

Cunha et al. haben eine Arbeit [CFRS12] veröffentlicht, die sich mit Spreadsheet Fehlern beschäftigt. In der Arbeit wird ein Fehlerkatalog definiert, validiert, anhand des dafür geschrieben SpreedSheet Detective evaluiert und daraufhin verbessert. Der Fehlerkatalog wurde in verschiedene Befunde unterteilt.

Unter die Kategorie der Typenbefunde fallen 2 Fehlermuster. Im ersten geht es um die Erkennung von leeren Zellen, bei denen vermutet wird, dass sie befüllt sein sollten. Das andere ist ein Pattern Finder und versucht Muster zu finden, indem eine Zelle aus dem Schema der anderen heraussticht. Diese beiden Fehlermuster sind zusammen ähnlich dem in Kapitel 5 vorgestellten Fehlermuster „Einer unter Anderen“. Da beide einen sehr ähnlichen Ansatz haben, halte ich es für sinnvoll diese auch in einem Muster zu vereinen.

In die Klasse der Inhaltsbefunde fällt das Fehlermuster namens „String Distance“. Es soll Tippfehler erkennen durch Verwendung der Wortdistanz. Dieses Fehlermuster wurde nachgebessert nach der Evaluation, indem Befunde die sich in nummerischen Werten unterscheidet, nicht berücksichtigt werden. Da die falsch-positiv Rate dadurch drastisch gesenkt wurde, wurde diese Verbesserung auch in unser Fehlermuster „Wortdistanz“ übernommen. Jedoch arbeitet das hier vorgestellte Fehlermuster mit einer Levenshtein-Distanz von 1 um Befunde zu melden. Da bei einem Buchstabendreher allerdings die Distanz 2 beträgt, wurde diese Distanz für einen Befund bei unserem Fehlermuster zusätzlich hinzugefügt.

In der Arbeit [HPD11] von Hermans et al. werden 5 Fehlermuster für Formeln erstellt und evaluiert. Im Gegensatz zu dieser Arbeit liegt der Ansatz jedoch darin qualitative und nicht quantitative Mängel aufzudecken. Qualitative Mängel führen zur Minderung der Qualität des Spreadsheets, was in der weiteren Verwendung beziehungsweise Veränderung des Spreadsheets zu quantitativen Fehlern führen kann. Da die Spreadsheetprüfung in der Industrie kaum Beachtung findet, sind Fehlermuster die Fehler die zu falschen Werten führen direkt aufdecken eher ein Anreiz ein Prüfwerkzeug einzusetzen.

5. Konzept

In diesem Kapitel werden nun fünf Konzepte vorgestellt und erläutert, die bei der Suche nach Fehlern in Spreadsheets helfen sollen.

5.1. Einer unter Anderen

5.1.1. Grundgedanke

Der Grundgedanke dieses Fehlermusters war es, Endnutzer dabei zu unterstützen fehlende Eingaben zu erkennen. Es sollte also eine leere Zelle gefunden werden, deren Umfeld darauf hindeutet, dass sie nicht leer sein sollte. So ein Ansatz wird auch in anderen Studien [CFRS12] verfolgt und dort wird gezeigt, dass dieses Fehlermuster in realen Spreadsheets Fehler aufdeckt. Allerdings wird in unserer Arbeit dieses Fehlermuster erweitert. Anstatt nur leere Zellen zwischen befüllten zu finden, lässt sich das Konzept auf alle Arten von Zellen erweitern. Das Fehlermuster soll eine Zelle finden, die vom Typ her nicht in ihr Umfeld passt. Dabei unterscheidet das Fehlermuster zwischen 3 Typen:

- Nummerische Werte
- Strings, die auch aus Zahlen bestehen können, aber keine rein nummerischen Werte sind
- leere Zellen.

Dadurch lassen sich Zellen finden, die ein gefundenes Shemata durchbrechen.

5.1.2. Konzept

Um dieses Fehlermuster umzusetzen, wird der Typ der Zelle bestimmt. Anhand der zu untersuchenden Zelle bilden wir entweder eine Horizontale, eine Vertikale oder ein Kreuz von dieser Zelle aus wie in Abbildung 2 dargestellt. Es werden als Standardwert jeweils 2 Zellen von der zu untersuchenden Zelle aus betrachtet, also fünf Zellen bei Linien und neun bei Kreuzen. Im Anschluss werden die Typen der Zellen im jetzt definierten Umfeld bestimmt. Wenn keine davon mit dem Typ der zu untersuchenden Zelle übereinstimmt, wird ein Befund registriert. Falls ein Zellentyp mit dem Typ der zu untersuchenden Zelle übereinstimmt, wird kein Befund angezeigt. Falls nun 2 falsche beziehungsweise leere Zellen nebeneinander liegen, erkennt das Fehlermuster dieses Fehlverhalten nicht.

Für die nicht-finanzorientierten Spreadsheets ist die Kreuzprüfung als Standard zu nutzen. Die Leserichtung von finanzorientierten Spreadsheets ist meist von oben nach unten, deshalb würde hier eine horizontale Prüfung ausreichen.

	A	B	C	D	E	F
1	**25,56 Euro**		**61,36 Euro**		**105 Euro**	
2	**Euro**	**Nachlass v.H.**	**Zahlungsbetrag Euro**	**Nachlass v.H.**	**Zahlungsbetrag Euro**	**Nachlass v.H.**
3	0	0	0	0	0	0
4	500	10	450	9	455	8
5	1000	13	870	11	890	9
6	1500	16	1260	13	1305	10
7	2000	19		15	1700	11,5
8	2500	21,5	1963	17	2075	12,5
9	3000	24,5	2265	19	2430	13,5
10	3500	27	2555	21	2765	15

Abbildung 2: Umfeld einer Zelle anhand einer Kreuzdefinition.

5.2. Referenzierung auf null-Werte

5.2.1. Grundgedanke

Fehler in Spreadsheets können eine falsche Formel als Grund haben, die wir statisch nicht überprüfen können, da wir ihren Hintergrund meist nicht kennen. Jedoch kann es auch einer falschen Referenzierung geschuldet sein. Der deutlichste und mit einem Fehlermuster gut zu überprüfende Fall ist es, wenn die Referenzierung auf eine leere Zelle Bezug nimmt.

5.2.2. Konzept

Um diese Fehlermustererkennung also zu realisieren, müssen wir alle Formeln des Spreadsheets anschauen. Aus diesen Formeln extrahieren wir zuerst alle Referenzierungen, welche in einer Liste gespeichert werden. Daraufhin müssen die Zellen, auf die Bezug genommen wird, einzeln angeschaut werden. Ist eine Zelle leer, wird die Zelle, die darauf verwiesen hat, mit einem Befund markiert.

5.3. Wortdistanz

5.3.1. Grundlage

Gerade kleinere Unternehmen verwalten ihre Inventarlisten unter anderem mit Hilfe von Spreadsheets. Bei manuellem Durchschauen solcher Spreadsheets ist ein Fehler aufgefallen, zu dem es sehr schnell kommen kann – ein Tippfehler. Auf den ersten Blick eventuell nur ein Schönheitsfehler. Unter Betrachtung, dass man Spreadsheets nach Worten filtern kann und somit dann Bestände, die zum Beispiel auf verschiedene Lager verteilt sind, zusammen zählen kann, führt dieser Tippfehler schnell zu falschen Werten.

	A	B	C
1	Artikel	Stückzahl	Lagerort
2			
3	MotorCclass	10	Tschechien
4	MotorCclass	20	Ungarn
5	MotorCclas	122	Deutschland
6	MotorCclass	10	China
7	MotorSprinter	34	Deutschland
8	MotorSprinter	12	China

Abbildung 3: Beispiel für Inventarliste mit einem Tippfehler

5.3.2. Konzept

Als naiven Ansatz könnte man das zu untersuchende Wort auf Gleichheit mit Wörtern aus dessen Umgebung kontrollieren. Falls das Wort jedoch nur einmal im Spreadsheet vorkommt, hätten wir einen falsch-positiven Befund. Abhilfe bei diesem Problem lässt sich jedoch durch den Levenshtein Algorithmus schaffen. Dieser Algorithmus untersucht wie viele Änderungen notwendig sind um aus einem Wort das andere zu machen. Bei diesem Fehlermuster werden also alle nicht-nummerischen Werte miteinander verglichen. Das Fehlermuster soll dann einen Befund melden, wenn die Wortdistanz des zu untersuchenden Strings zu einem anderen eins beträgt und es keinen weiteren String gibt, zudem die Wortdistanz null ist. Es soll also ein Befund gemeldet werden, wenn es keinen gleichen String gibt, aber einen, der durch eine Änderung zu dem untersuchten String gemacht werden kann. Im Beispiel aus Abbildung 3 würde also nur bei A5 ein Befund gemeldet werden, da sie einen Abstand von 1 zu A3, A4 und A6 hat und es keinen String mit dem Abstand 0 gibt. A3 hat auch einen Abstand von 1 zu A5 jedoch auch den Abstand null zu A4 und A6. Eine andere Studie [CFRS12] zeigt, dass wenn man Strings ignoriert, die sich nur durch Zahlen unterscheiden, man die falsch-positiv Rate um über 80% senken kann.

5.4. Separate Mehrfachreferenzierung in Formeln

5.4.1. Grundlage

So gut wie jede Spreadsheet-Software unterstützt ihren Benutzer bei der Erstellung von Funktionen. So lassen sich beim Erstellen einer SUMME()-Formel meist die zu addierenden Zellen im Spreadsheet einfach anklicken. Durch das Klicken kann es dazu kommen, das eine Zelle ungewollt doppelt referenziert wird indem man ungenau beziehungsweise wiederholt klickt ohne es zu realisieren. Dies ist einer der Gründe warum es zu ungewollten Mehrfachreferenzierung kommt. Berichte[EuSpRIG], die auf Grundlage von Untersuchungen der EuSpRIG entstanden sind, zeigen, dass solche Fehler möglicherweise zu schweren (finanziellen) Auswirkungen führen können.

Um die falsch-positiv Rate bei diesem Fehlermuster zu senken, muss darauf geachtet werden, dass alle als beabsichtigte Mehrfachreferenzierungen erkannten Referenzierungen nicht als Befund gemeldet werden.

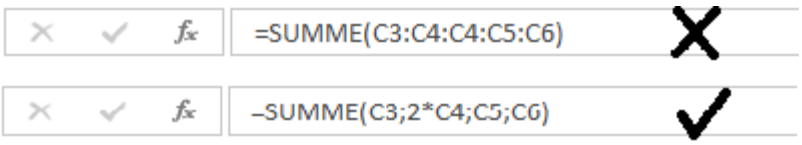

Abbildung 4: Beispiel für Ungewollte(oben) und gewollte(unten) Mehrfachreferenzierungen.

5.4.2. Konzept

Für dieses Fehlermuster schaut man sich jede Zelle mit Formel an. Besitzt sie eine Funktion mit Referenzen oder besteht rein aus Additionen oder Subtraktionen von Referenzen wird kontrolliert, ob eine Referenz innerhalb dieser Konstrukte mehrmals separat aufgeführt ist. Ist eine mehrfache Referenzierung so dargestellt wie in Abbildung 4 unten, wird das als gewollte Mehrfachreferenzierung angesehen und nicht als Befund gemeldet.

5.5. Eingaben an nicht berücksichtigten Stellen

5.5.1. Grundlage

Das folgende Fehlermuster umschließt zwei verschiede Fehlverhalten des Erstellers beziehungsweise Endnutzers. Zum einen soll es erkennen, wenn der Nutzer aufgrund von Schwierigkeiten bei der Auswahl der Eingabezelle den Wert in eine falsche Zelle einträgt. Dies kann bei schlechter Übersichtlichkeit des Spreadsheets passieren, falls zum Beispiel die Beschriftung der Zellen weit von der Eingabezelle entfernt ist. Zum anderen dient es als Überprüfung von falsch gesetzten Referenzen und spielt zum Teil dadurch den Gegenspieler zur Referenzierung auf leere Zellen. Wenn laut Beschriftung diese Zelle für eine bestimmte Eingabe vorgesehen ist, ihr Wert jedoch nie abgerufen wird, deutet es darauf hin, dass die Referenz entweder falsch gesetzt wurde oder unbeabsichtigt überhaupt nicht gesetzt wurde.

5.5.2. Konzept

Um dieses Fehlermuster zu überprüfen suchen wir uns alle Zellen heraus, die keine Berechnungen enthalten. Auch Referenzen der Form „=B12“ werden hier außen vorgelassen, da diese oftmals zur Übersichtlichkeit beziehungsweise Verdeutlichung verwendet werden. Wir suchen also alle Zellen mit ausschließlich Konstanten. Daraufhin durchsuchen wir alle Zellen auf Referenzen. An den Stellen an der die Liste der Referenzen nicht mit der Liste der Zellen mit Konstante übereinstimmt, wird ein Befund auf Seiten der Zelle mit konstanten Werten gemeldet. Somit erhalten wir alle Zelle die nur Konstante enthalten, die aber nicht in irgendeiner Form referenziert werden.

6. Umsetzung

Das Spreadsheet Inspection Framework bietet die Möglichkeit weitere Fehlermuster für die statische Prüfung zu implementieren.

Der grundsätzliche Ansatz muss hierbei für jedes Fehlermuster vorgenommen werden (<name> steht dabei für den Namen des Fehlermusters):

In der Klasse sif.model.policy.policyrule.implementations.<name>PolicyRule.java werden zum einen alle konfigurierbaren Werte erstellt und Informationen zu dem Fehlermuster wie Name des Autors, Name des Fehlermusters und eine Beschreibung des Fehlermusters gegeben.

In der Klasse sif.model.violations.single.<name>SingleViolation.java wird der Verstoß mit verschiedene Werten beschrieben. Als zu implentierendes Interface wird ISingleVialoation benutzt. Für eine spätere Ausgabe sind das verursachende Element, das Fehlermuster, das den Befund gefunden hat und ein String, der den Befund beschriebt, abrufbar. Außerdem wird die schwere des Fehlers berechnet und der Wert lässt sich durch die Funktion getWeightedSeverityValue() abrufen.

Nachdem die Rahmenbedingungen implementiert wurden, wird in der Klasse sif.technicalDepartment.equipment.testing.facilities.implementations.<name>TestFacility .java die Erkennung für das Fehlermuster implementiert. Die Klasse erbt dabei die Funktion violationList run() von der MonolithicTestFacility. Diese prüft das Spreadsheet nach einem bestimmten Muster bestimmte Zellen und sammelt alle Verstöße in einer ViolationList, die zurückgegeben wird. Sie benutzt dabei die konfigurierbaren Werte aus sif.model.policy.policyrule.implementations. <name>PolicyRule.java.

Es folgt die genauere Beschreibung der Umsetzung für das jeweilige Fehlermuster. Dabei wird kurz auf die konfigurierbaren Werte eingegangen und dann das Muster, dass zur Erkennung des Fehlermuster ausgeführt wird und die Funktionen, die dabei genutzt werden genauer beschrieben.

6.1. Einer unter Anderen

Dieses Fehlermuster soll in der Klasse OneAmongOthersPolicyRule.java realisiert werden. Ihre konfigurierbaren Werte sind dabei, die Form der Umgebungsbestimmung (Horizontale/ Vertikale /Kreuz), die Größe der Umgebung und eine Liste der zu ignorierenden Zellen.

Es wird über alle Zellen iteriert, die nicht als zu ignorierend markiert sind. Bei der Betrachtung der Zelle wird zuerst ihr Typ bestimmt (Double, String, null) Daraufhin wird je nach der zuvor ausgewählten Form der Umgebung diese abgefragt und den Typ des Inhalts bestimmt. Ist in der vorgegebenen Umgebung keine Zelle, die den gleichen Typ hat wie die zu untersuchende Zelle, wird diese mit einem Befund bei der Befundsliste registriert.

6.2. Referenzierung auf null-Werte

Dieses Fehlermuster wurde in der Klasse RefToNullPolicyRule.java realisiert. Ihr konfigurierbarer Wert ist eine Liste mit den nicht zu untersuchenden Zellen.

Es wird jede Zelle, die nicht als zu ignorieren markiert ist und die Formeln beinhaltet, betrachtet. Aus diesen Formeln werden alle „Tokens“ extrahiert. Dabei werden die „Tokens“ überprüft und geschaut, ob sie Referenzen enthalten. Für jede Referenz wird die Funktion isRefNull(reference) aufgerufen, die die Zelle kontrolliert auf die Bezug genommen wird und diese auf einen Wert überprüft. Ist dieser Wert „null“ wird ein Fehler zu der Fehlerliste hinzugefügt. Als verursachendes Element wird dabei die Referenz übergeben. Wenn dieses Vorgehen für alle Zelle abgeschlossen ist, wird die Liste mit Verstößen übergeben.

6.3. Wortdistanz

Dieses Fehlermuster wurde in der Klasse StringDistancePolicyRule.java realisiert. Ihr konfigurierbarer Wert ist die maximale Levenshtein Distanz und einer Liste mit Zellen, die nicht untersucht werden sollen.

In 2 verschachtelten For-Schleifen wird jeder String mit jedem anderen String des Spreadsheets verglichen. Dazu wird die Funktion getDistance(sourceString, testString) (vergleiche Abbildung 5) aufgerufen, welche den Levenshtein Algorithmus implementiert und die Distanz der Wörter zueinander berechnet. Ist dieser Wert kleiner als der zuvor bestimmte kleinste Abstand wird er mit dem neuen Wert überschrieben. Ist am Ende der Wert kleiner als die konfigurierte maximale Distanz und größer als 0. Wird ein Befund für die Zelle registriert und nach vollständigem Durchlauf in Form einer Befundsliste zurückgegeben.

```
int prevCost[] = new int[sourceLength+1];
int actCost[] = new int[sourceLength+1];
int swapCost[];
// indizes
int sourceIndex;
int testIndex;
// char in testString at position testIndex-1
char testIndexChar;

int cost;

for (sourceIndex = 0; sourceIndex<=sourceLength; sourceIndex++) {
    prevCost[sourceIndex] = sourceIndex;
}

for (testIndex = 1; testIndex<=testLength; testIndex++) {
    testIndexChar = testString.charAt(testIndex-1);
    actCost[0] = testIndex;

    for (sourceIndex=1; sourceIndex<=sourceLength; sourceIndex++) {
        // cost = 0|1
        cost = sourceString.charAt(sourceIndex-1)==testIndexChar ? 0 : 1;
        // minimum of cell to the left + 1, to the top + 1,
        //diagonally left and up + cost
        actCost[sourceIndex] = Math.min(Math.min(actCost[sourceIndex-1]+1, prevCost[sourceIndex]+1),  prevCost[sourceIndex-1]+cost);
    }
    // actual distance counts to previous distance counts
    swapCost = prevCost;
    prevCost = actCost;
    actCost = swapCost;
}
return prevCost[sourceLength];
```

Abbildung 5: Hauptberechnungen des Levenshtein Algorithmus

6.4. Separate Mehrfachreferenzierung in Formeln

Dieses Fehlermuster soll in der Klasse MultipleSameRefPolicyRule.java realisiert werden. Ihr konfigurierbarer Wert ist die Liste der zu ignorierenden Zellen.

Das Muster soll die ausgehenden Referenzen jeder Zelle einzeln anschauen. Wird dabei innerhalb der ArrayListe, die diese Referenzen speichert, eine Zelle doppelt genannt, wird ein Befund auf der Befundsliste registriert.

Ob dieser Ansatz funktioniert konnte nicht bestätigt werden, da nicht sicher ist wie der Parser die in Kapitel 5.4 gezeigten akzeptierten Mehrfachreferenzierungen behandelt.

6.5. Eingaben an nicht berücksichtigten Stellen

Dieses Fehlermuster soll in der Klasse NonConsideredValuesPolicyRule.java realisiert werden. Ihr konfigurierbarer Wert ist eine Liste der Zellen, die nicht betrachtet werden sollen.

Es sollen alle Zellen angeschaut werden und geschaut werden, ob ihr Inhalt aus einer Konstanten besteht. Alle Zellen, die dieses Merkmal haben werden weiter untersucht. Dabei wird die ArrayList der eingehenden Referenzen betrachtet. Ist die Liste leer, zeigt dies das auf diesen konstanten Wert von nirgendwo referenziert wird und deshalb wird die Zelle dann bei der Befundsliste gemeldet.

7. Zusammenfassung und Ausblick

7.1. Zusammenfassung

Anhand von Studien wurde gezeigt, dass über 80% der in der Industrie verwendete Spreadsheets Fehler enthalten und kommerzielle Prüfwerkzeuge in der Industrie auf Grund von konzeptionellen Einschränkungen kaum verwendet werden. Um das Erkennungspotential des Spreadsheet Inspection Framework zu steigern wurden im Verlauf dieser Arbeit anhand von manuell gefunden Fehlern und Berichten über Fehler in Spreadsheets 5 Fehlermuster erarbeitet und konzipiert.

Die konzipierten Fehlermuster „Referenzierung von null-Werten“ und „Wortdistanz“ wurden in das Spreadsheet Inspection Framework implementiert. Die bei Erstellung der Ausarbeitung nicht implementierten Fehlermuster wurden auf Grundlage des Programmcodes beschrieben um eine Implementierung zu vereinfachen. Diese Fehlermuster erweitern die Fehlererkennung des Spreadsheet Inspection Frameworks und steigern dadurch dessen praktischen Nutzen. Jedoch erkennt das Spreadsheet Inspection Framework nach der Implementierung der verbleibenden konzipierten Fehlermuster nur einen Bruchteil von Fehlermustern, die in Spreadsheets vorhanden sind.

7.2. Ausblick

In zukünftigen Arbeiten könnte das Tool um weitere statische Fehlermuster erweitert werden, da die vorhandenen 8 Fehlermuster nur eine überschaubare Anzahl an Fehlern findet, jedoch viele Fehlermuster unentdeckt bleiben. Neben quantitativen könnten qualitative Fehlermuster entwickelt werden, um die Spreadsheet-Qualität zu erhöhen und das Spreadsheet dadurch weniger anfällig für spätere Fehler zu machen.

Des Weiteren könnte die statische Prüfung besser in das SIFEI eingebunden werden. Dabei sollte die Auswahl, der zur Verfügung stehenden Fehlermuster auswählbar sein und die Konfigurierung der Parameter des Fehlermusters direkt über SIFEI verwaltet werden können.

Die Benutzerfreundlichkeit des Spreadsheet Inspection Frameworks beziehungsweise des SIFEIs könnte anhand von automatischen Prüfungen und einer Internationalisierung gesteigert werden.

A. Anhang

A.1. Inhalt der CD

Diese Arbeit ist auf der beigelegten CD dokumentiert. Sie beinhaltet dabei folgenden Inhalt:

- Implementierung
- Die Ausarbeitung im PDF-Format
- Die Zusammenfassung und das Summary als Plain Text

Literaturverzeichnis

[Butler00] R. J. Butler. „Is This Spreadsheet a Tax Evader? How H.M. Customs and Excise Test Spreadsheet Applications". In Proceedings of the 33rd Hawaii International Conference on System Sciences, 2000

[CHM08] M. Clermont, C. Hanin, R. T. Mittermeir. „A Spreadsheet Auditing Tool Evaluated in an Industrial Context." CoRR, abs/0805.1741, 2008.

[CFMMS12] J. Cunha, J. P. Fernandes, P. Martins, J. Mendes, J. Saraiva. „SmellSheet Detective: A Tool for Detecting Bad Smells in Spreadsheets". In Proceedings of the 2012 IEEE Symposium on Visual Language and Human-Centric Computing, 2008.

[CFRS12] J. Cunha, J. P. Fernandes, H. Ribeiro, J. Saraiva. „Towards a Catalog of Spreadsheet Smells". In The 12th International Conference on Computational Science and Its Applications, 2012.

[Doust13] E. Doust. „Visualisierung von Fehlern in Spreadsheets". Bachelor Thesis, University of Stuttgart, 2013.

[EuSpRIG] http://www.eusprig.org/horror-stories.htm FH1202, FH1217

[GHJJR96] D.F. Galletta, K.S. Hartzel, S.E.Johnson, J. L. Joseph, S. Rustagi. „Spreadsheet presentation and error detection: an experimental study". In Proceedings of the 29th Annual Hawaii International Conference on System Sciences, 1996.

[HPD11] F. Hermans, M. Pinzger, A. van Deursen. „Detecting Code Smells in Spreadsheet Formulas". In Proceeding of the 34rd international conference on Software engineering, 2011.

[IEEE09] IEEE Standard Classification for Software Anomalies 1044-2009, 2009.

[LL04] R. J. Lawrence, J. Lee. „Financial Modelling of Project Financing Transactions," In Institute of Actuaries of Australia Financial Services Forum, August, 2004.

[LuLi07] J. Ludewig, H. Lichter. „Software Engineering - Grundlagen, Menschen, Prozesse, Techniken". dpunkt.verlag, 2007.

[Lemke12] M. Lemke. „Dynamische Prüfung von Spreadsheets". Diploma Thesis, University of Stuttgart, 2013.

[Panko07] R. R. Panko. „Recommended Practices for Spreadsheet Testing".

[Panko08] R. R. Panko. „Revisiting the Panko-Halverson Taxonomy of Spreadsheet Errors". CoRR,abs/0809.3613, 2008.

[Panko09] R. R. Panko. „Applying Code Inspection to Spreadsheet Testing". p.32f.

[Panko96] R. R. Panko, R. H. Halverson. „Spreadsheets on Trial: A Framework for Research on Spreadsheet Risks". In Proceedings of the 29th Annual Hawaii International Conference on System Sciences, 1996.

[Panko98] R. R. Panko. „What we know about spreadsheet errors". In Journal of End User Computing's, 1998.

[PKLC13] P. Poon, F. Kuo, H. Liu, T. Y. Chen. „How can non-technical end users effectively test their spreadsheets?". In Information Technology and People, 2013.

[RD04] F. van Rysselberghe, S. Demeyer. „Evaluating clone detection techniques from a refactoring perspective". In Proceedings of the 19th International Conference on Automated Software Engineering, 2004.

[Scheu14] J. Scheurich. „Benutzerschnittstelle für einen Spreadsheet-Prüfstand". Bachelor Thesis, 2014.

[TO80] R. N. Taylor, L. J. Osterweil. „Anomaly detection in concurrent software by static data flow analysis". 1980.

[Zitze12] S. Zitzelsberger. „Error Detection in Spreadsheets". Diploma Thesis, University of Stuttgart, 2012.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten

_____

Ort, Datum, Unterschrift